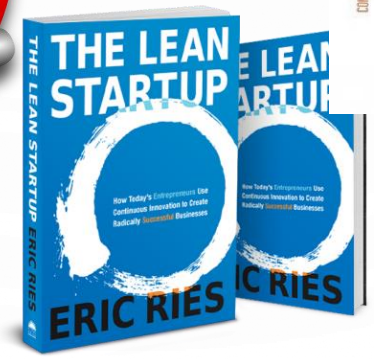
TEST DRIVEN DEVELOPMENT

AXP Academy

DRUMIA OVIDIU – TRAINER BIO



AXP Academy

DRUMIA OVIDIU – TRAINER BIO

- 6 years – Java Developer
- Oracle Certified Professional, Java SE Programmer 6
- 5 years – Agile Coaching & Consulting
- Agile Project Management – Foundations (w/ Arie van Bennekum)
- Facilitating Agile (w/ Arie van Bennekum)
- Certified Business Executive Coach
- XP software development Experience (as dev) – Telenet Project

AXP Academy

YOUR TURN!

- Name & Team
- Favorite food & favorite restaurant?
- Please create some name tags out of A4s

TDD KNOWLEDGE

RATE YOUR KNOWLEDGE OF THE FOLLOWING TOPICS ON A SCALE OF 1 TO 5 (5 IS THE HIGHEST)

TDD KNOWLEDGE

Topic	Rating (1-5)
Test case	
Test-first	
Test-last	
System Under Test (SUT)	
Test Double (Dummy/Stub/Mock/Fake/Spy)	
Fixture/Setup/Teardown	
3 Laws of TDD	
Regression	
Refactoring	
Test suite	
XP Practices	
Whitebox Testing/Blackbox testing	

Rate your knowledge of the following topics on a scale of 1 to 5

5 is the highest

OBJECTIVES

- Master TDD concepts
- Master TDD mindset
- Master TDD practice

AXP Academy

BEFORE WE BEGIN

TRAINER: OVIDIU DRUMIA

ORGANIZATIONAL ASPECTS

- Schedule
- Presence
- Breaks

AGENDA

- Why TDD?
  - Software development facts and statistics
  - XP practices overview
- What is TDD?
  - Test types
  - Test case
  - System under test
- How can we do TDD?
  - 3 laws of TDD
  - Red, Green, Refactor
  - Test first
  - TDD mistakes
  - Test double(s) - tomorrow
  - Regression - tomorrow
  - Refactoring - tomorrow

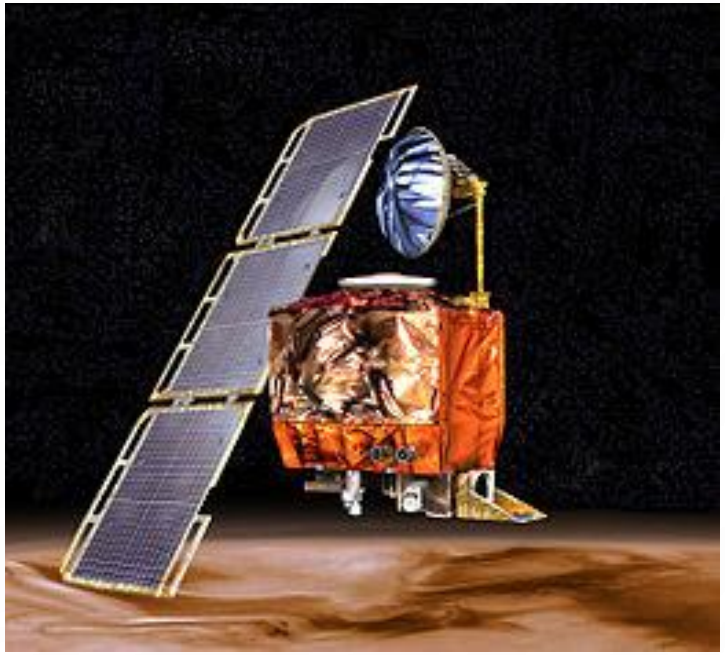
AXP Academy

WHY TDD?

SOFTWARE BUGS ARE MOSTLY EMBARRASSING (MY TOP 3 LIST)

1. Mars Climate Orbiter (1988)

Turns out that forgetting to convert miles into km….



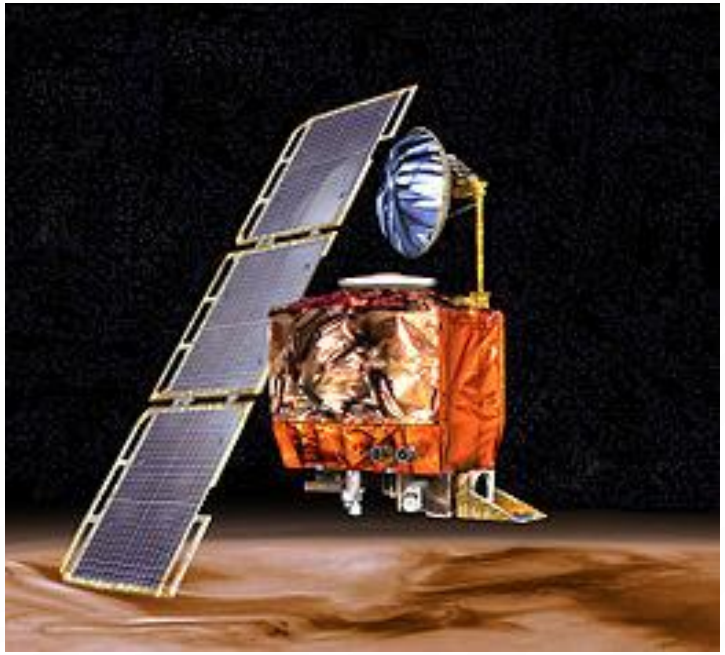
Source: https://en.wikipedia.org/wiki/Mars_Climate_Orbiter

AXP Academy

SOFTWARE BUGS ARE MOSTLY EMBARRASSING (MY TOP 3 LIST)

1. Mars Climate Orbiter (1988)

…leads you to actually lose a robotic space probe worth $125m



AXP Academy

SOFTWARE BUGS ARE MOSTLY EMBARRASSING (MY TOP 3 LIST)

2. Vancuver Stock Exchange (1982)

Your stock might be performing poorly or…

Source: http://hubpages.com/education/Propagation-of-Rounding-Errors-in-Solving-Math-Problems

AXP Academy

SOFTWARE BUGS ARE MOSTLY EMBARRASSING (MY TOP 3 LIST)

2. Vancuver Stock Exchange (1982)

…the system might be truncating your

rating instead of rounding it.

(500 point loss, in 2 years)

AXP Academy

SOFTWARE BUGS ARE MOSTLY EMBARRASSING (MY TOP 3 LIST)

3. IBM’s Deep Blue beats Kasparov (1997)

Chess genius AI or…

Source: http://www.wired.com/2012/09/deep-blue-computer-bug/

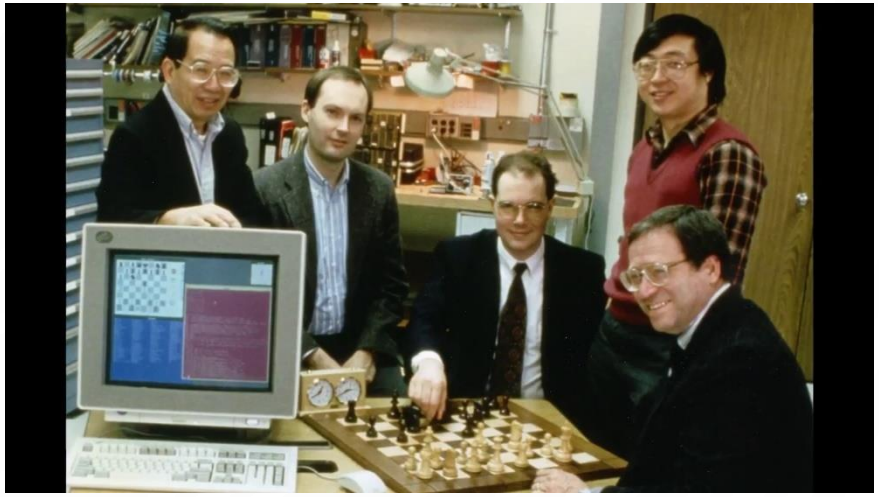
AXP Academy

SOFTWARE BUGS ARE MOSTLY EMBARRASSING (MY TOP 3 LIST)

3. IBM's Deep Blue beats Kasparov (1997)

…flaw which resulted in a completely random move,

that confused the opponent?



AXP Academy

CONCERNES

- Writing tests AND writing code takes more time than writing code
- TDD will not validate our system
- This TDD business is only for small bugs – we don’t have small bugs here
- TDD is not for large scale projects

TDD IS NOT FOR LARGE SCALE PROJECTS

- All large solutions don’t just materialize out of nowhere
- They are created in modest steps (think in small units)
- NASA has been experimenting with TDD since 1960

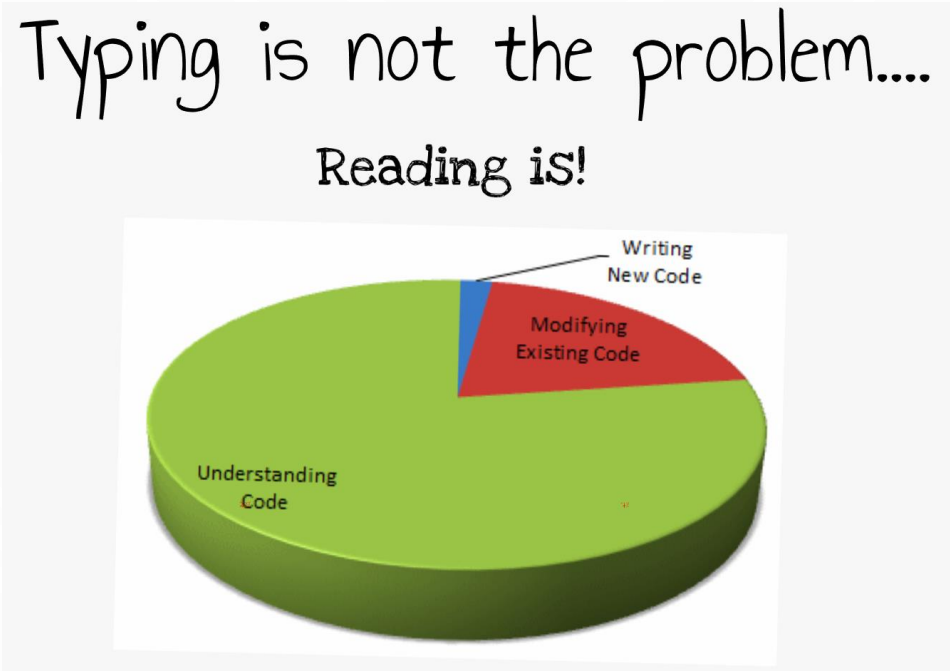
TDD IS SLOW

- I have to write 1 line of test for each line of code, this means a TDDer writes twice as much code for the same functionality!
- Guess what?

TDD IS SLOW

- Typing is NOT the problem!
- Running the debugger to see what was supposed to happen, but didn't (63%)
- Setting up things just right so that the debug session repeats the bug in production (20%)
- Sitting with someone else to show them in the debugger why their code does not work (7%)
- Reading log files (10%)

TDD IS SLOW


Typing is not the problem....
Reading is!
Writing New Code
Modifying Existing Code
Understanding Code

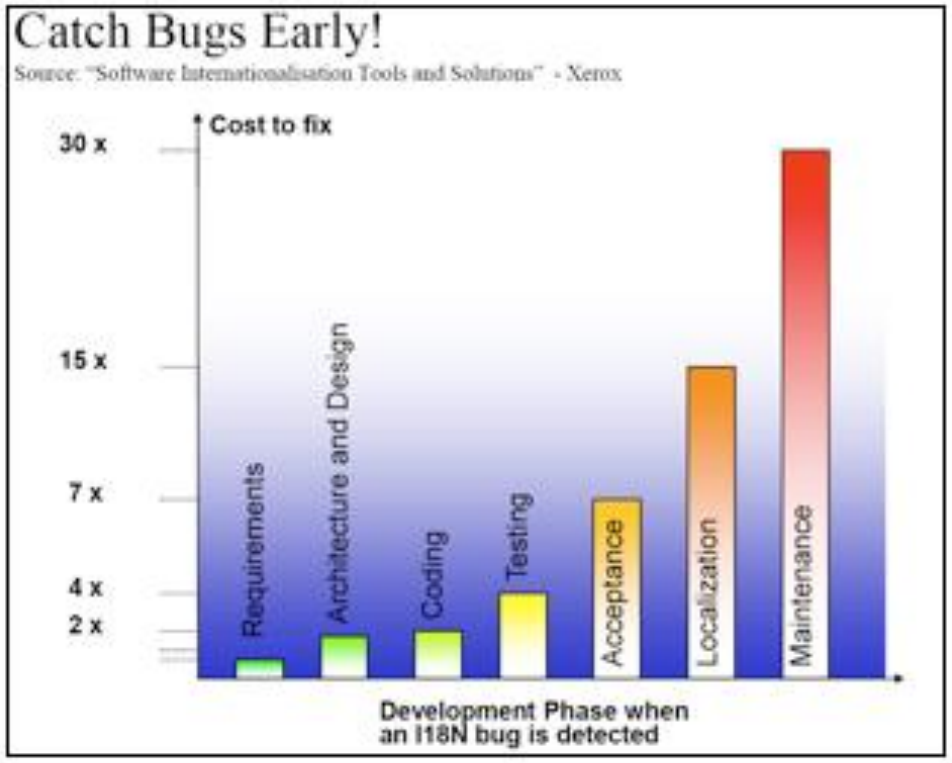
IT ALL BOILS DOWN TO…

- A software defect is a result of an object not doing what it’s expected to
- Testing each object to the extent of it’s responsibilities, helps reduce these effects
- TDD != no bugs
- A bug = a missing test or wrong test

101 TO FIX A BUG

- Reproduce it
- Frame it
- Write a test to reproduce it and thus fails
- Fix the test

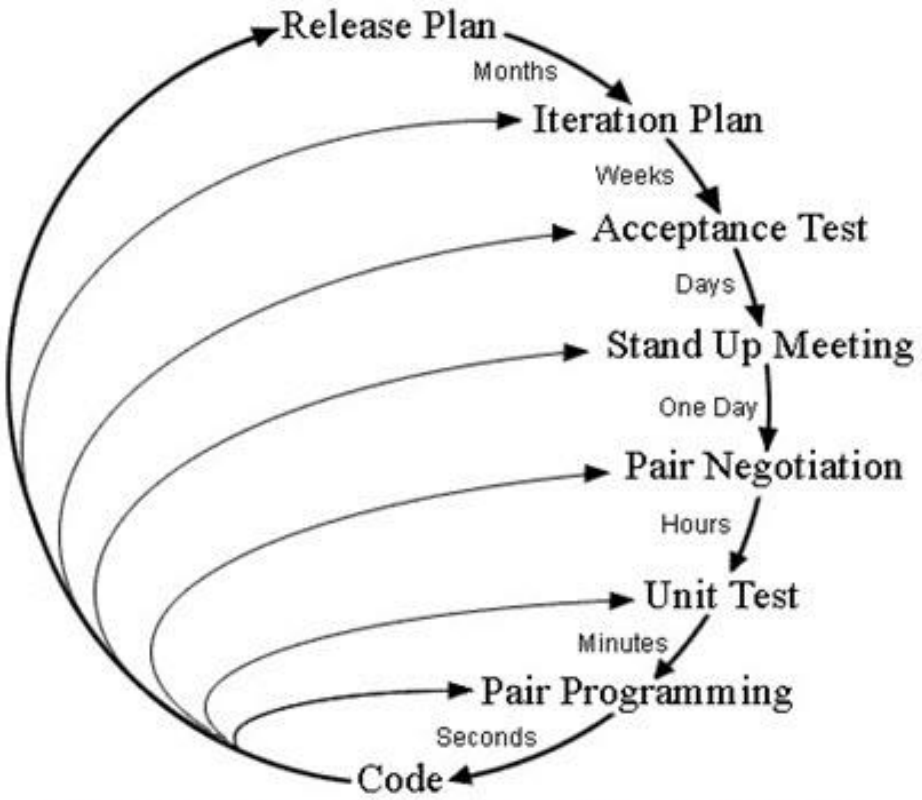
LATE BUGS ARE KILLING US!



It's far more efficient to find and fix i18n issues at the source level, rather than depending upon testing and localization iterations.

AXP Academy

FEEDBACK LOOP



Typical Planning & Feedback Loop of Agile XP Model

AXP Academy

BENEFITS

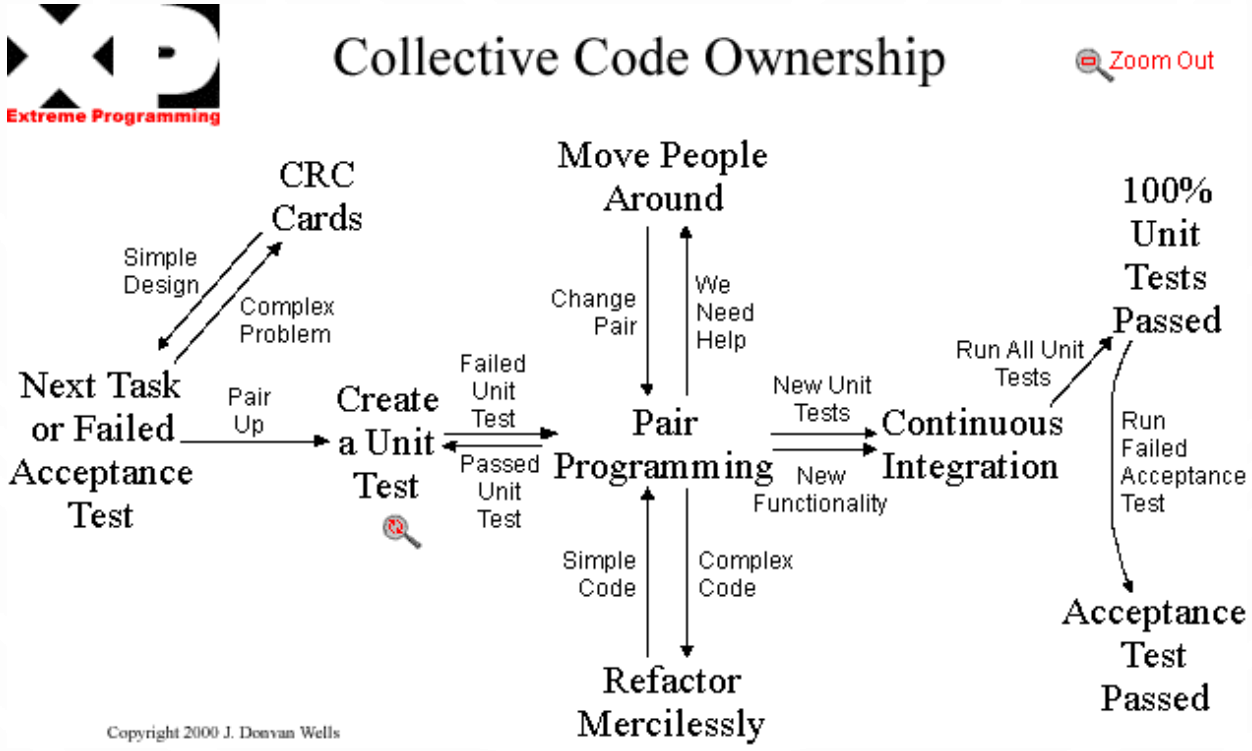
- Less legacy code
  - Legacy code = code without tests (Michael Feathers)
- Separation of concerns
- Less debugging
- Easier debugging
- Code is easier to maintain and extend
- You stop writing code that won’t be used
- Total code implementation time is actually shorter
- Live documentation
  - A code example that shows you what your code does

FOR ME?

- “Always code as if the guy that ends up maintaining the your code will be a violent psychopath who knows where you live.” (Martin Golding)
- “Programming is like sex: one mistake and you will be providing support for a lifetime.” (Michael Sinz)
- “I like to change and refactor my code without fear!” (Ovidiu)

AXP Academy

EXTREME PROGRAMMING PRACTICES



AXP Academy

ACCOUNT DESIGN SESSION

- “As a bank customer I want to be able to make withdrawals and deposits from my account.” – User story
- Split in teams of max 5 people
- What should the “Account” object look like?

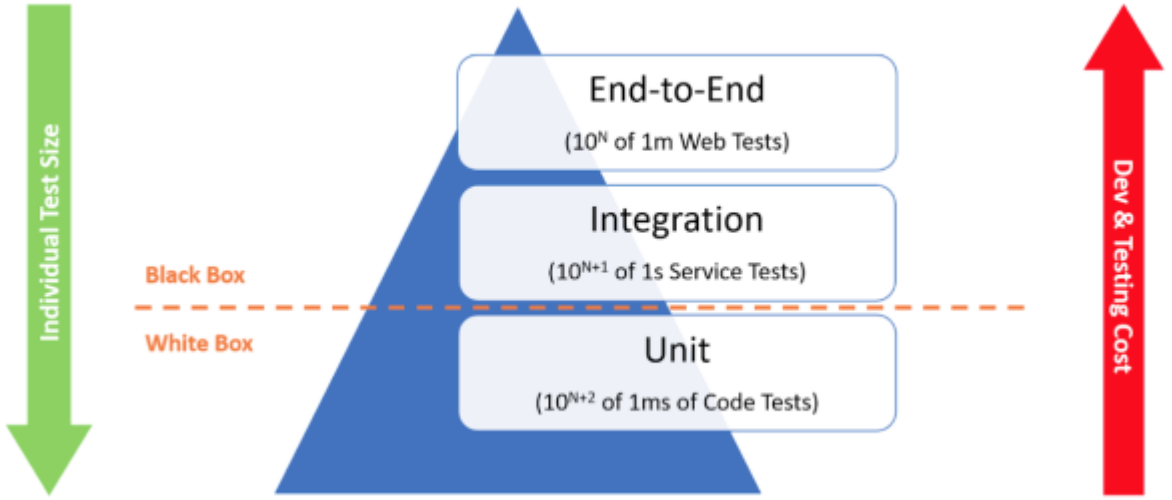
WHAT IS TDD?

AXP Academy

TYPES OF TESTS

- Unit testing
- Integration Testing
- End 2 End Testing

TESTING PYRAMID



AXP Academy

TEST CASE

- Def: A **TEST CASE** is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

Input > System under test > Output

If Output == Expected -> pass

Else Fail

GERKIN LANGUAGE

- Given – Context/Initial Scenario
- When – Action which is performed
- Then – Expected Result

SYSTEM UNDER TEST

- DEF: **System under test** (**SUT**) refers to a system that is being tested for correct operation.

TEST CASE

- (In Pairs) Write as many test cases as possible for your “Account” object
  - 10 - 15 min

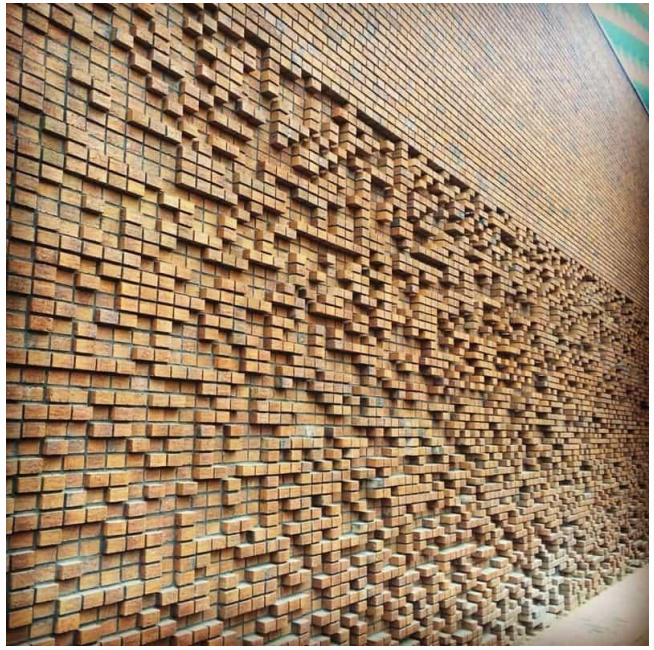
AXP Academy

TEST FIRST VS TEST LAST

- What are the differences you notice in the two approaches?

TDD IS

- Building the system test case, by test case



DEMO

DELAYED

AXP Academy

DEMO – DELAY
(HTTPS://GITHUB.COM/OVIDIUDRUMIA/DELAY)

- Scenario 1: Airplane not late
  - Print: “All right!”
- Scenario 2: Airplane is under 5 min late
  - Print: “Ok..”
- Scenario 3: Airplane is under 15 min late
  - Print: “Finally..”
- Scenario 4: Airplane is more than 30 min late
  - World explodes!

AXP Academy

HOW CAN WE DO TDD?

TEST STRUCTURE

```
@Test
public void getMinutesDelayed_givenAirplaneAndMinutesDelayed_correctMinutesDelayedReturned(){
    Airplane airplane = new Airplane( minutes: 2);

    int minutesDelayed = airplane.getMinutesDelayed();

    assertThat(minutesDelayed).isEqualTo(2);
}
```

Unit test naming convention

Setup (Given)

Exercise (When)

Verify (Then)

AXP Academy

FIXTURE & TEARDOWN

```
@BeforeClass
public static void setUpClass() {
  System.out.println("@BeforeClass setUpClass");
  myExpensiveManagedResource = new ExpensiveManagedResource();
}

@AfterClass
public static void tearDownClass() throws IOException {
  System.out.println("@AfterClass tearDownClass");
  myExpensiveManagedResource.close();
  myExpensiveManagedResource = null;
}
```

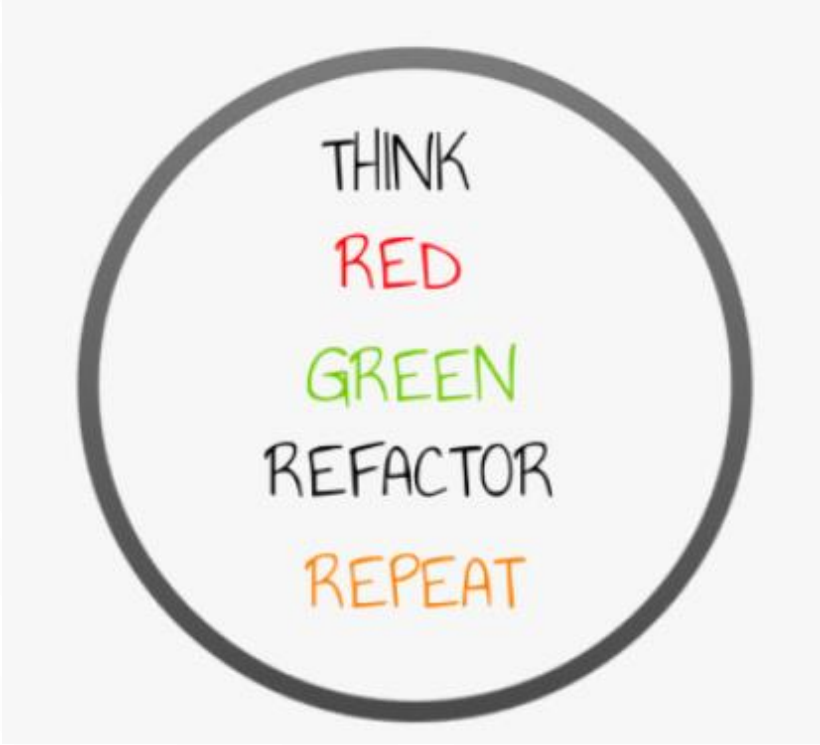
```
@Before
public void setUp() {
  this.println("@Before setUp");
  this.myManagedResource = new ManagedResource();
}

@After
public void tearDown() throws IOException {
  this.println("@After tearDown");
  this.myManagedResource.close();
  this.myManagedResource = null;
}
```

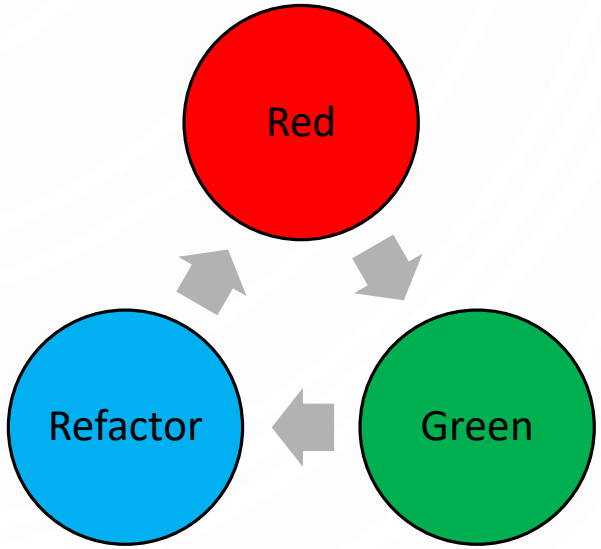

AXP Academy

3 LAWS OF TDD (UNCLE BOB)

- You are not allowed to write any production code unless it is to make a failing unit test pass.
- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
- You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

ROUTINE





AXP Academy

ROUTINE

THINK

Think about what tests you're writing.
Don't write empty tests.

TAKE YOUR TIME

RED

Write a very small amount of code.
This should break your build.
(1 minute)

GREEN

Write only enough code to fix the test.
Don't worry about it being pretty.
(30 seconds)

REFACTOR

Refactor your code without fear.
Improve the look, remove smells.
After each change, run your test and make sure it still passes.

REPEAT

Do the whole cycle again.
You should be repeating this cycle many times an hour.
(20 - 40)

AXP Academy

WHAT DEFINES GOOD TESTS?

- Fast: tests should run fast!
- Independent: they should not depend on each other
- Repeatable: they should be repeatable in any environment
- Self-validating: RED or GREEN
- Timely: write test before production code

but above all

READABLE

AXP Academy

TDD MISTAKES

1. No test at the beginning
   a) Skipping writing test first “sometimes” when it’s “convenient”
   b) Skipping writing test first mentally (I know exactly what my next 20 steps are)
2. Skipping RED phase
   a) Failing test does not compile
   b) Test has not been run before creating the implementation that satisfies it
   c) Test is broken (it fails because of a different reason than it has been designed)
3. Skipping the Refactor phase
   a) “code should read like a well-written novel” (Uncle Bob)
   b) self-commented: “comments are always failures” (Uncle Bob)
4. Too large steps
   a) Difficult initialization (large “given” section)
   b) Too many assertions (large “then” section)
5. Goal: Code Coverage
   a) Do not create tests to gain 100% coverage
   b) High coverage does not guarantee the functionality is well tested

LABS – DAY 1

AXP Academy

EXERCISE WARM UP – SPACEBOOK

(HTTPS://GITHUB.COM/OVIDIUDRUMIA/SPACEBOOK)

- Scenario 1: Person has a username
- Scenario 2: Person has a gender
- Scenario 3: Person has an age
- Scenario 4: Person has a list of friends

AXP Academy

EXERCISE – SPACEBOOK

(HTTPS://GITHUB.COM/OVIDIUDRUMIA/SPACEBOOK)

- User Story 1:
  - A **person** has a **username** and a list of **friends**.
  - Username cannot be null, empty or spaces
- User Story 2:
  - Becoming a friend means adding a bidirectional relationship.
  - Adding yourself should not be possible
- User Story 3:
  - A person can **receive messages** from a friend only.
  - A message has a date, sender and body (text)
  - You can ask a person for all received messages
- User Story 4:
  - You can ask a person for all received messages, **sorted** by date.
- User Story 5:
  - You can ask a person for received messages from a friend, sorted by date.

AXP Academy

DAY 1 - RECAP

- WHY TDD?
- What is TDD?
- How do we do TDD?

AXP Academy

DAY 2

BUG FIX – DRACULA

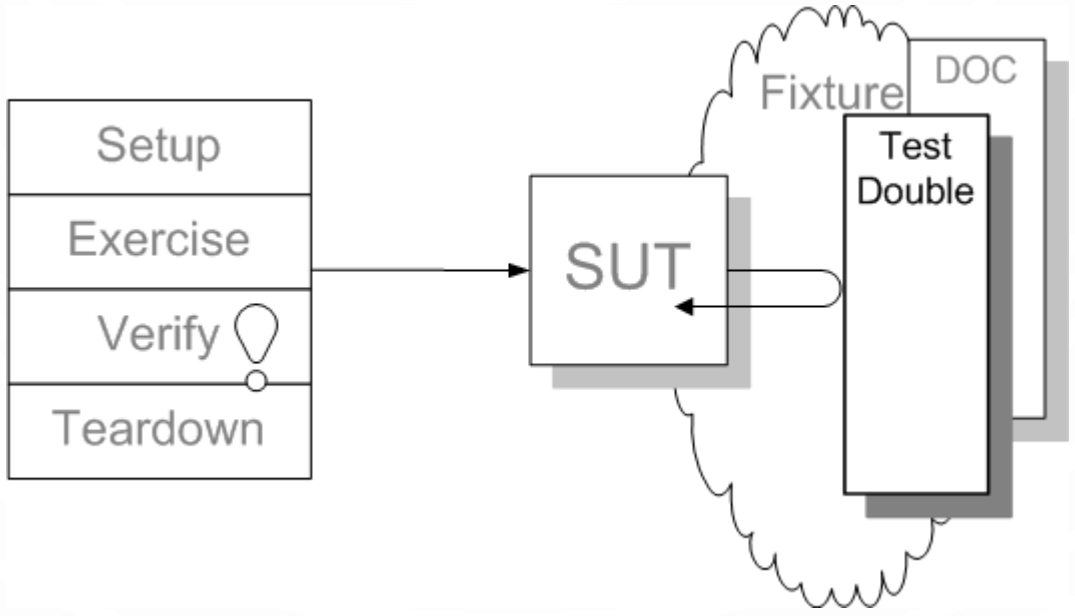
HTTPS://GITHUB.COM/OVIDIUDRUMIA/DRACULA

- User Story:
  - A hunter can go hunting vampires between midnight and 6 am
- Bug:
  - Hunter can go hunting on midnight
  - EXPECTED: Can go hunting
  - ACTUAL: Cannot go hunting

AXP Academy

TEST DOUBLES

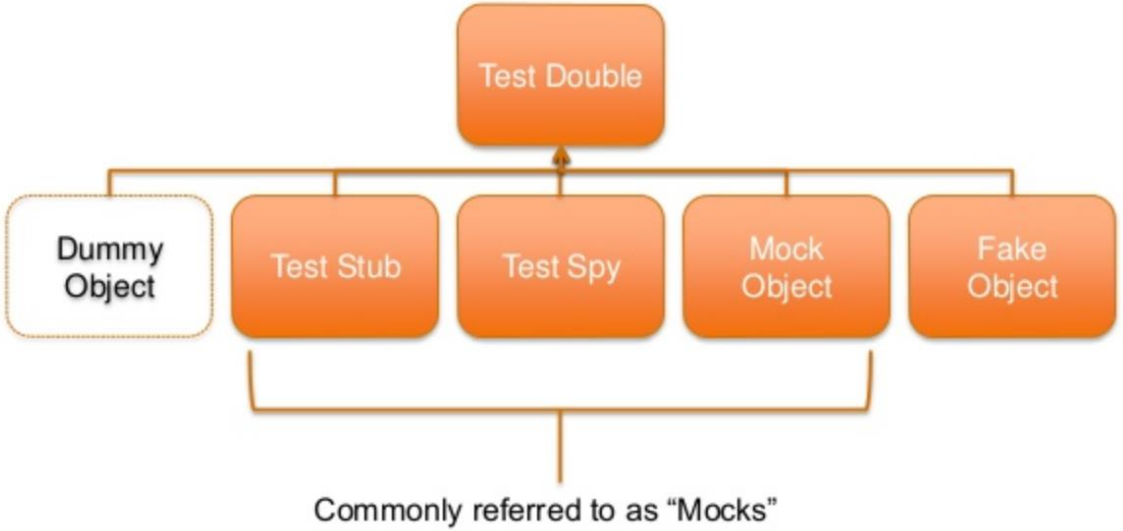
- Test Double is a generic term for any case where you replace a production object for testing purposes.



AXP Academy

TEST DOUBLES

- Test Double is a generic term for any case where you replace a production object for testing purposes.



AXP Academy

TEST DOUBLES DUMMY

- **Dummy** objects are passed around but never actually used. Usually they are just used to fill parameter lists.

```
interface Authorizer {
  public Boolean authorize(String username, String password);
} >_An interface._
```

```
public class DummyAuthorizer implements Authorizer {
  public Boolean authorize(String username, String password) {
        return null;
          }
} >_That's a Dummy._
```

TEST DOUBLES DUMMY

- **Dummy** objects are passed around but never actually used. Usually they are just used to fill parameter lists.

```
public class System {
	public System(Authorizer authorizer) {
		this.authorizer = authorizer;
	}

	public int loginCount() {
		//returns number of logged in users.
	}
}

@Test
public void newlyCreatedSystem_hasNoLoggedInUsers() {
	System system = new System(new DummyAuthorizer());
	assertThat(system.loginCount(), is(0));
}
```

```
public class System {
	public System(Authorizer authorizer) {
		this.authorizer = authorizer;
	}

	public int loginCount() {
		//returns number of logged in users.
	}
}

@Test
public void newlyCreatedSystem_hasNoLoggedInUsers() {
	System system = new System(new DummyAuthorizer());
	assertThat(system.loginCount(), is(0));
}
```

TEST DOUBLES STUB

- **Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.

```
public class AcceptingAuthorizerStub implements Authorizer {
  public Boolean authorize(String username, String password) {
        return true;
          }
}
```

TEST DOUBLES SPY

- **Spies** are stubs that also record some information based on how they were called. We could implement a counter telling us how many times authorizations were made using this service.

```
public class AcceptingAuthorizerSpy implements Authorizer {
  public boolean authorizeWasCalled = false;

  public Boolean authorize(String username, String password) {
        authorizeWasCalled = true;
        return true;
  }
}
```

TEST DOUBLES MOCK

- **Mocks** are pre-programmed with expectations which form a specification of the calls they are expected to receive. They can throw an exception if they receive a call they don't expect and are checked during verification to ensure they got all the calls they were expecting.

```
public class AcceptingAuthorizerVerificationMock implements Authorizer {
  public boolean authorizeWasCalled = false;

  public Boolean authorize(String username, String password) {
        authorizeWasCalled = true;
        return true;
  }

  public boolean verify() {
        return authorizedWasCalled;
  }
}
```

TEST DOUBLES FAKE

- **Fake** objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an InMemoryTestDatabase is a good example).

```
public class AcceptingAuthorizerFake implements Authorizer {
        public Boolean authorize(String username, String password) {
              return username.equals("Bob");
        }
}
```

IN PRACTICE - MOCKITO

MOCKITO – INITIALIZING TEST

```
@RunWith(MockitoJUnitRunner.class)
public class MockitoAnnotationTest {
    ...
}
```

```
@Before
public void init() {
    MockitoAnnotations.initMocks(this);
}
```

MOCKITO – SETTING UP A MOCK

```java
@Test
public void whenNotUseMockAnnotation_thenCorrect() {
    List mockList = Mockito.mock(ArrayList.class);

    mockList.add("one");
    Mockito.verify(mockList).add("one");
    assertEquals(0, mockList.size());

    Mockito.when(mockList.size()).thenReturn(100);
    assertEquals(100, mockList.size());
}
```

```java
@Mock
List<String> mockedList;

@Test
public void whenUseMockAnnotation_thenMockIsInjected() {
    mockedList.add("one");
    Mockito.verify(mockedList).add("one");
    assertEquals(0, mockedList.size());

    Mockito.when(mockedList.size()).thenReturn(100);
    assertEquals(100, mockedList.size());
}
```

MOCKITO – SETTING UP A SPY

```java
@Test
public void whenNotUseSpyAnnotation_thenCorrect() {
    List<String> spyList = Mockito.spy(new ArrayList<String>());

    spyList.add("one");
    spyList.add("two");

    Mockito.verify(spyList).add("one");
    Mockito.verify(spyList).add("two");

    assertEquals(2, spyList.size());

    Mockito.doReturn(100).when(spyList).size();
    assertEquals(100, spyList.size());
}
```

```java
@Spy
List<String> spiedList = new ArrayList<String>();

@Test
public void whenUseSpyAnnotation_thenSpyIsInjected() {
    spiedList.add("one");
    spiedList.add("two");

    Mockito.verify(spiedList).add("one");
    Mockito.verify(spiedList).add("two");

    assertEquals(2, spiedList.size());

    Mockito.doReturn(100).when(spiedList).size();
    assertEquals(100, spiedList.size());
}
```

MOCKITO – SETTING UP A SPY

```java
@Test
public void whenNotUseSpyAnnotation_thenCorrect() {
    List<String> spyList = Mockito.spy(new ArrayList<String>());

    spyList.add("one");
    spyList.add("two");

    Mockito.verify(spyList).add("one");
    Mockito.verify(spyList).add("two");

    assertEquals(2, spyList.size());

    Mockito.doReturn(100).when(spyList).size();
    assertEquals(100, spyList.size());
}
```

```java
@Spy
List<String> spiedList = new ArrayList<String>();

@Test
public void whenUseSpyAnnotation_thenSpyIsInjected() {
    spiedList.add("one");
    spiedList.add("two");

    Mockito.verify(spiedList).add("one");
    Mockito.verify(spiedList).add("two");

    assertEquals(2, spiedList.size());

    Mockito.doReturn(100).when(spiedList).size();
    assertEquals(100, spiedList.size());
}
```

MOCKITO – CAPTOR

```java
@Test
public void whenNotUseCaptorAnnotation_thenCorrect() {
    List mockList = Mockito.mock(List.class);
    ArgumentCaptor<String> arg = ArgumentCaptor.forClass(String.class);

    mockList.add("one");
    Mockito.verify(mockList).add(arg.capture());

    assertEquals("one", arg.getValue());
}
```

```java
ArgumentCaptor<Person> argument = ArgumentCaptor.forClass(Person.class);
verify(mock).doSomething(argument.capture());
assertEquals("John", argument.getValue().getName());
```

```java
@Mock
List mockedList;

@Captor
ArgumentCaptor argCaptor;

@Test
public void whenUseCaptorAnnotation_thenTheSam() {
    mockedList.add("one");
    Mockito.verify(mockedList).add(argCaptor.capture());

    assertEquals("one", argCaptor.getValue());
}
```

MOCKITO – INJECT MOCKS

```java
@Mock
Map<String, String> wordMap;

@InjectMocks
MyDictionary dic = new MyDictionary();

@Test
public void whenUseInjectMocksAnnotation_thenCorrect() {
    Mockito.when(wordMap.get("aWord")).thenReturn("aMeaning");

    assertEquals("aMeaning", dic.getMeaning("aWord"));
}
```

```java
public class MyDictionary {
    Map<String, String> wordMap;

    public MyDictionary() {
        wordMap = new HashMap<String, String>();
    }
    public void add(final String word, final String meaning) {
        wordMap.put(word, meaning);
    }
    public String getMeaning(final String word) {
        return wordMap.get(word);
    }
}
```


AXP Academy

MOCKITO – NPE USING MOCKS

```
1  public class NPETest {
2
3      @Mock
4      List mockedList;
5
6      @Test
7      public void test() {
8          Mockito.when(mockedList.size()).thenReturn(1);
9      }
10 }
```

- Same behavior using @Spy

LAB 3 – PET SHOP

HTTPS://GITHUB.COM/OVIDIUDRUMIA/PETSHOP

- User Story 1:
  - A PetShop has a stock. Items can be added
- User Story 2:
  - When a new item is added, an email is sent with the name of the item

AXP Academy

LAB 4 – FLIGHT

HTTPS://GITHUB.COM/OVIDIUDRUMIA/FLIGHT/

- User Story 1:
  - A Flight has a list of Passengers
  - A Flight has an Id (int)
  - Passengers only have a name
- User Story 2:
  - You can add Passengers to a Flight
  - You can get the Number of Passengers on a Flight
  - You can check if a Passenger is on a Flight
- User Story 3:
  - A flight has a Maximum Number of Seats
  - When the flight is full, adding a Passenger causes an exception
- User Story 4:
  - You can use a FlightBookingService to book a seat on a Flight using a FlightId and a person
  - Booking a Seat means adding the Passenger to the Flight
  - If the Flight with the given Id is not found an exception is thrown
  - A Flight is loaded from a database (customer is not sure of which DB to use)

AXP Academy

LAB 5 – MOVIE RENTAL

HTTPS://GITHUB.COM/OVIDIUDRUMIA/MOVIE

- Refactor the code using the catalog (http://refactoring.com/catalog/)
- User Story 1:
- Add a new type of movie (Adult)
- Price is € 5 per day
- Think SOLID
  - S – SRP – Single Reponsibility Principle
  - O – OCP – Open/Closed Principle
  - L – LSP – Liskov Substitution Principle
  - I - ISP - Interface Segregation Principle
  - D – DSP – Depencency Inversion Principle

AXP Academy

DAY 2 - RECAP

AXP Academy

TDD KNOWLEDGE

Topic	Rating (1-5)
Test case	
Test-first	
Test-last	
System Under Test (SUT)	
Test Double (Dummy/Stub/Mock/Fake/Spy)	
Fixture/Setup/Teardown	
3 Laws of TDD	
Regression	
Refactoring	
Test suite	
XP Practices	
Whitebox Testing/Blackbox testing	

Rate your knowledge of the following topics on a scale of 1 to 5

5 is the highest

THANK YOU!

AXP Academy

BIBLIOGRAPHY

- Test Driven Development, Kent Beck
- Refactoring, Martin Fowler
- Clean Code, Robert C. Martin
- Working With Legacy Code, Michael Feathers
- https://code.tutsplus.com/articles/tdd-terminology-simplified--net-30626
- https://www.future-processing.pl/blog/tdd-tests/
- https://en.wikipedia.org/wiki/Test-driven_development
- https://github.com/junit-team/junit4/wiki/test-fixtures
- https://martinfowler.com/bliki/TestDouble.html
- https://www.baeldung.com/mockito-annotations

AXP Academy

- Email me:
  - Ovidiu.drumia@axpconsulting.com
- Add me on LikendIn
  - https://www.linkedin.com/in/ovidiu-drumia-27621418/

AXP Academy